一种求解特征值问题的广义共轭梯度算法*

张 宁[†], 李 瑜[‡], 谢和虎[§], 徐 然[¶], 游春光[‖]

摘要

本文基于阻尼块反幂法与子空间投影算法设计了一种求解特征值问题的广义共轭梯度算法, 同时也实现了相应的计算软件包. 然后对算法和计算过程进行了一系列的优化来提高算法的稳定性、计算效率和并行可扩展性, 使得本文的算法适合在并行计算环境下求解大规模稀疏矩阵的特征值. 所形成的软件包是基于Matrix-Free和Vector-Free设计的, 可以应用于任意的矩阵向量结构. 针对几种典型矩阵的测试结果表明本文的算法和软件包不但具有良好的数值稳定性, 同时相比于SLEPc软件包中的LOBPCG以及Jacobi-Davidson解法器有2-6倍的效率提升. 软件包的网址: `https://github.com/pase2017/GCGE-1.0`.

关键词: 特征值问题; 阻尼块反幂法; 广义共轭梯度法; 稳定性; 可拓展性.

A Generalized Conjugate Gradient Method for Eigenvalue Problems

Abstract

A generalized conjugate gradient method is proposed to solve eigenvalue problems. This method is designed by combining the dumping block inverse power scheme, subspace

*受科学挑战项目(No.TZ2016002), 国家自然科学基金(11771434, 91730302, 91330202)项目资助.

†中国科学院数学与系统科学研究院, 计算数学研究所, 国家数学与交叉科学中心, 科学与工程计算国家重点实验室, 中国、北京 100190; 中国科学院大学, 数学科学学院, 北京 100049

‡天津财经大学, 管理可计算建模协同创新中心, 中国、天津 300222

§中国科学院数学与系统科学研究院, 计算数学研究所, 国家数学与交叉科学中心, 科学与工程计算国家重点实验室, 中国、北京 100190; 中国科学院大学, 数学科学学院, 北京 100049

¶北京应用物理与计算数学研究所, 中国、北京 100088

‖中物院高性能数值数值模拟软件中心, 中国、北京 100088

projection method. Furthermore, based on the properties of the proposed method, a series of optimization techniques is developed to improve the stability, computing efficiency and scalability. We also introduce a computing package GCGE (Generalized Conjugate Gradient Eigensolver) which is developed based on the proposed method here. Some numerical examples are provided to validate the stability, computing efficiency and scalability of the method in this paper. The corresponding computing package can be downloaded from the web site: `https://github.com/pase2017/GCGE-1.0`.

Keyword: eigenvalue problem; dumping block inverse power; generalized conjugate gradient; stability; scalability.

MR. 65N30, 65B99.

目 录

List of Tables

List of Figures

1 算法介绍

本文介绍一种基于阻尼块反幂法的并行特征值求解算法及其相应的高效实现方法. 阻尼块反幂法是反幂法、阻尼思想和子空间投影方法的组合, 同时考虑具体的并行实现效率来设计出相应的代数特征值求解算法. 这里设计的算法主要是用来求解对称正定矩阵最小端特征值及其相应的特征向量. 当然本文算法设计的思想也可以用来构造求解非对称与非正定矩阵特征值问题的方法. 在具体设计的时候主要是利用阻尼块反幂法结合并行计算的特点来设计高效的特征值求解算法.

本文主要考虑求解如下的广义代数特征值问题

$$Ax = \lambda Bx.$$

为了简单起见, 这里规定矩阵A和B是对称正定的. 现有比较流行的特征值解法器一般都基于Krylov子空间算法[15], 比如Arnoldi算法、LOBPCG算法和Jacobi-Davidson算法等. 基于Krylov子空间的算法(Arnoldi算法和Lanczos算法[15])的特征值解法器, 需要精确求解线性方程组, 否则得不到相应的上Hessenberg矩阵. 对于一些条件数很大的矩阵, 使用迭代法难以收敛, 而使用直接法求解, 会使得求解时并行效率较差. 基于LOBPCG或者Jacobi-Davidson等算法的特征值解法器, 不需要精确求解线性方程组, 因此即使对于条件数很大的矩阵, 也可以使用迭代法求解, 并行效率比较好. 但目前流行的特征值解法器中, LOBPCG稳定性较差, 这是由其产生子空间的方式以及正交化的方式所导致的, 使得LOBPCG在求解一些条件数较大或者精度要求较高的特征值问题时往往求解失败. 而Jacobi-Davidson算法需要求解不定且几乎奇异的线性方程组, 需要专门设计相应的线性解法器来进行求解. 在使用Jacobi-Davidson解法器的时候需要设置的参数较多, 往往难以选择最优的计算参数而使得计算效率较低. 求解大规模特征值问题的并行算法应满足如下条件: 一方面该算法不需要精确求解其中所包含的线性方程组, 另一方面该算法需要具有稳定、高效的特点才能适合求解大规模矩阵特征值问题的要求. 同时为了提高算法的并行可扩展性, 尽量把算法的主要部分转化成具有良好可扩展性的计算方式.

基于上面的考虑, 本文的第一个目的是设计一个稳定、高效、高可扩展性的特征值算法. 为了适应在并行机上求解大规模矩阵的特征值问题, 此算法不需要精确求解其中所包含的线性方程组, 并且同时具有稳定、高效、高扩展性的特点. 我们基于阻尼块反幂法和子空间投影算法的思想设计一种求解特征值问题的广义共轭梯度(Generalized Conjugate Gradient, 简称GCG)算法. 与LOBPCG算法相比较, 主要的区别在于计算向量组W的方式, GCG利用反幂法结合共轭梯度(CG)迭代的方式, 而LOBPCG方法利用的是计算残差向量组的方式[5, 8, 11, 12, 13, 16]. 我们知道随着特征对精度的增加, 残差将会变得越来越小从而使得LOBPCG在数值上不稳定[5], 而反幂法没有这种缺点. 当然在没有任何机器误差的情况下这两种方式是等价的[5, 17].

本文的第二个目的是基于上面设计的特征值算法及其性质设计一个具有比较好的数值稳定性(鲁棒性)和可扩展性的特征值并行解法器. 由于算法主要是基于矩阵和向量的整体操作来运行的, 所以这里的解法器可以设计成Matrix-Free和Vector-Free的形式, 也就是用户可以基于自己的矩阵和向量格式及矩阵和向量操作来产生具体的特征值解法器. 同时为了提高解法器的效率, 本文也将介绍一些基于算法性质的优化设计.

本文接下来的安排为: 本节主要介绍GCG算法, 第2节介绍基于GCG算法的软件包, 第3节介绍对GCG算法的具体实现及其改进, 第4节将进行一些数值实验来验证本文提出算法的计算效率和并行效率. 最后一节给出对本文的总结.

1.1 GCG算法

GCG算法是一种子空间迭代算法, 它用阻尼块反幂法迭代的方式来生成一个三元向量组$[X,P,W]$, 并以之张成求解特征值问题的子空间. 三元向量组$[X,P,W]$中的X是本次迭代中的近似特征向量, P是本次迭代中的近似特征向量减去上次迭代中近似特征方向的分量, W是对X进行一步不精确反幂法迭代得到的向量. 由于向量组中的W是通过一步不精确的反幂法迭代产生的, 对于对称正定问题, 我们常采用一定步数的CG迭代来得到, 因此这个算法命名为广义共轭梯度算法(Generalized Conjugate Gradient Method). 假设要求解`nev`个特征值, 算法1给出了具体的GCG算法过程.

Algorithm 1 GCG算法

1. 初始选取X, $P=[\,]$, $W=[\,]$, 其中X中有`nev`个向量. 计算小规模特征值问题 $X^TAXC=X^TBXC\Lambda$得到特征值Λ和特征向量C. 更新$X=XC$.
2. 对线性方程组$W=A^{-1}(BX\Lambda)$以X为初始值执行一定的CG迭代步得到向量组W.
3. 定义空间$V=[X,P,W]$, 对其进行正交化得到关于矩阵B的单位正交基组V.
4. 计算Rayleigh-Ritz问题: $V^TAVC=V^TBVC\Lambda$, 利用得到的特征向量C和Ritz值Λ获得新的特征向量逼近$X_{\texttt{new}}=VC$和相应的特征值逼近.
5. 检查$X_{\texttt{new}}$的收敛性, 如果收敛的个数等于`nev`, 计算结束.
6. 否则, 计算$P=X_{\texttt{new}}\backslash X$, 更新$X=X_{\texttt{new}}$, 回到步2进行下一次迭代.

注 1. *算法1 步骤6中的$X_{\texttt{new}}\backslash X$表示在$X_{\texttt{new}}$中去掉$X$方向上的分量, 后文有对具体实现方式的描述.*

算法1与LOBPCG算法的区别主要在于生成向量W的方式和对向量组V的正交化方式. 算法1采用块反幂法的方式来产生W, 同时为了确保算法的稳定性, 这里对向量组V进行完全的正交化. LOBPCG方法中的向量组W是当前特征逼近的残差向量, 并

且X, P, W各自自身进行正交化, 并不对整体向量组V进行正交化. 首先当近似特征向量具有一定精度的时候, 残差向量将会变得很小而带来数值不稳定性. 另外由于没有进行整体正交化使得V具有线性相关性而导致在计算Rayleigh-Ritz问题的时候算法终止[5, 8]. 这也就是在算法1中我们要对向量组V进行正交化的原因, 并且我们后面会根据算法的性质来提高完全正交化的计算效率.

在本文之后的部分, 我们将针对算法1进行一系列具体实现的考虑和效率的优化, 使得算法在保持稳定性的同时具有更高的计算效率和并行效率, 以及可以进行非正定矩阵特征值问题的计算, 主要包括分批计算技巧、块计算的方式、块正交化、快速计算Rayleigh-Ritz问题等技术, 其目的是为了提高计算效率和算法的并行可扩展性.

1.2 特征值分批计算

算法1中求解Rayleigh-Ritz问题的时候, 计算特征值的部分是串行的, 这一部分的计算时间很难通过增加进程数来减少. 当计算的特征值个数较多时, 由于子空间特征值计算时间与特征值个数是超线性关系, 这样就造成总的计算时间随着特征值个数的增长是超线性的. 基于这样的考虑, 算法的构造应该尽量减少子空间特征值问题的维数. 当需要计算的特征值个数较多时, 我们将特征值进行分批计算. 每次只计算k个P向量和k个W向量. 算法过程见算法2.

Algorithm 2 GCG算法-特征值分批计算

1. 初始选取$[X_1, X_2]$, $X_0 = [\,]$, $P = [\,]$, $W = [\,]$, 其中X_1中有k个向量, X_0 包含已经收敛的特征向量组. 定义向量组$\widetilde{X} = [X_0, X_1, X_2]$. 计算特征值问题 $\widetilde{X}^T A \widetilde{X} C = \widetilde{X}^T B \widetilde{X} C \Lambda$. 更新$[X_1, X_2] = \widetilde{X} C$.
2. 对线性方程组$W = A^{-1}(BX_1\Lambda)$以X_1为初始值执行一定的CG迭代步得到向量组W.
3. 定义空间$V = [X_0, X_1, X_2, P, W]$, 并计算Rayleigh-Ritz问题: $V^T AVC = V^T BVC\Lambda$, 获得新的特征向量逼近$[X_1^{\mathtt{new}}, X_2^{\mathtt{new}}]$和相应的特征值逼近.
4. 检查收敛性: 更新$[X_c, \widetilde{X}_1, \widetilde{X}_2] = [X_1^{\mathtt{new}}, X_2^{\mathtt{new}}]$, $X_0 = [X_0, X_c]$, 其中X_c为本次迭代新收敛的向量组, $\widetilde{X}_1$为未收敛向量的前k个, 其余为$\widetilde{X}_2$. 记$X_1^{\mathtt{old}}$ 为$[X_1, X_2]$中与$\widetilde{X}_1$对应的部分.
5. 如果X_0中的向量个数等于`nev`, 计算结束.
6. 否则, 计算$P = \widetilde{X}_1 - X_1^{\mathtt{old}}$, 记$[X_1, X_2] = [\widetilde{X}_1, \widetilde{X}_2]$, 回到步2 进行下一次迭代.

上面对Rayleigh-Ritz问题求解的处理也是本文与文章[5]中的区别之一, 这里只需求解Rayleigh-Ritz问题的目标特征对, 而不需要求解全部的特征对. 通过上面的处理, 我们可以进一步减少求解Rayleigh-Ritz问题的时间在总计算时间中所占的比率. 由

于Rayleigh-Ritz问题是一个稠密矩阵的特征值问题, 并行计算很难发挥出计算潜力, 需要特别的考虑, 在后面的小节3.3和3.4中有进一步的讨论.

1.3 带位移的GCG算法

由于在算法2的第2步中计算W向量组时默认使用CG迭代方法, 而CG迭代方法只适用于对称正定线性方程组的求解. 为了求解对称不定矩阵的特征值问题, 我们对算法进行了位移处理, 使得算法也可以处理非正定的问题. 当然对于更广泛的情形, 未来也会考虑MINRES迭代方法和GMRES迭代方法.

我们基于算法2进行修改, 同样考虑对特征值进行分批计算, 且每次只计算k个P向量和k个W向量. 算法过程调整如下, 其中位移的更新方式表示使用位移后最大特征值为最小特征值的100倍, 这样既能保证特征值是正的, 又能使得第一个特征值不会太小.

Algorithm 3 带位移的GCG算法

1. 初始选取$[X_1, X_2]$, $X_0 = [\,]$, $P = [\,]$, $W = [\,]$, 其中X_1中有k个向量, X_0 包含已经收敛的特征向量组. 定义向量组$\widetilde{X} = [X_0, X_1, X_2]$. 令位移参数$\texttt{shift} = 0$. 计算特征值问题$\widetilde{X}^T A \widetilde{X} C = \widetilde{X}^T B \widetilde{X} C \Lambda$. 更新$[X_1, X_2] = \widetilde{X} C$.
2. 对线性方程组$W = (A + \texttt{shift} \cdot B)^{-1}(B X_1 \Lambda)$以$X_1$为初始值执行一定的CG迭代步得到新的向量组$W$.
3. 定义空间$V = [X_0, X_1, X_2, P, W]$, 并计算Rayleigh-Ritz问题:
$$V^T (A + \texttt{shift} \cdot B) V C = V^T B V C \Lambda,$$
获得新的特征向量逼近$[X_1^{\texttt{new}}, X_2^{\texttt{new}}]$和相应的特征值逼近, 并将特征值减去$\texttt{shift}$以得到原问题的近似特征值.
4. 检查收敛性: 更新$[X_c, \widetilde{X}_1, \widetilde{X}_2] = [X_1^{\texttt{new}}, X_2^{\texttt{new}}]$, $X_0 = [X_0, X_c]$, 其中X_c为这次迭代新收敛的向量组, $\widetilde{X}_1$为未收敛向量的前k个, 其余为$\widetilde{X}_2$. 记$X_1^{\texttt{old}}$ 为$[X_1, X_2]$中与$\widetilde{X}_1$对应的部分. 更新$\texttt{shift} = (\lambda_{\texttt{nev}} - 100 \cdot \lambda_1)/99$.
5. 如果X_0中的向量个数等于$\texttt{nev}$, 计算结束.
6. 否则, 计算$P = \widetilde{X}_1 - X_1^{\texttt{old}}$, 记$[X_1, X_2] = [\widetilde{X}_1, \widetilde{X}_2]$, 回到步2进行下一次迭代.

2 软件包介绍

基于上节介绍的GCG算法, 我们构造了求解大规模矩阵特征值问题的广义共轭梯度解法器(Generalized Conjugate Gradient Eigensolver, 简称GCGE)软件包, 这个软件包

是Matrix-Free和Vector-Free的. 在该软件包中, 我们提供了基于CSR (串行行压缩稀疏矩阵), Hypre [10], PASE, PETSc [4], PHG [20], 以及SLEPc [9]软件包中的矩阵结构、向量结构以及矩阵向量操作的特征值解法器的调用方式. 如果用户使用的是其中一种矩阵向量结构, 就可以直接使用软件包中的例子程序来求解特征值问题. 此外, 该软件包支持用户提供的任意矩阵结构、向量结构以及矩阵向量操作. 用户只需参照已有的调用方式提供自己的矩阵结构、向量结构、矩阵向量的基本操作, 以及相应的接口函数, 就可以使用GCGE软件包在自己的矩阵向量结构下求解矩阵特征值问题. 具体的软件包C语言代码可以在GitHub上找到(网址:`https://github.com/pase2017/GCGE-1.0`), 欢迎下载并测试与使用.

2.1 程序结构

GCGE软件包主要分为gcge, app, example三个部分. 其中, gcge为GCGE的主体部分, 用来具体实现GCG 算法, 这部分是Matrix-Free和Vector-Free的, 不涉及任何具体的矩阵结构、向量结构与矩阵向量操作; app提供具体的矩阵向量结构与矩阵向量操作下对gcge的调用接口, 目前提供了CSR, Hypre, PASE, PETSc, PHG以及SLEPc等软件包的调用接口; example中给出了相应的测试和调用的例子.

以SLEPc格式的矩阵向量结构为例, 头文件gcge.h中包含了GCGE算法求解代数特征值问题的所有调用函数以及参数设置函数; app文件夹中的头文件gcge_app_slepc.h中提供了SLEPc格式的矩阵向量结构对gcge的调用接口. 用户使用SLEPc格式的矩阵向量结构计算矩阵特征值时, 只需要包含gcge.h与gcge_app_slepc.h这两个头文件就可以调用GCGE软件包中提供的函数进行矩阵特征值求解的操作.

2.1.1 内核结构

头文件gcge.h中包含了GCG算法求解代数特征值问题的所有调用函数以及参数设置函数. 该文件中包含了gcge_type.h, gcge_ops.h, gcge_para.h, gcge_workspace.h, gcge_rayleighritz.h, gcge_orthogonalize.h, gcge_xpw.h, gcge_eigsol.h, gcge_solver.h等头文件, 他们共同构成了实现GCG算法的基础.

第一层也就是最底层的头文件gcge_type.h中定义了一些基本类型和预定义, 这个头文件会被gcge中的所有其他头文件所包含.

第二层含有两个头文件gcge_ops.h和gcge_para.h. 其中在文件gcge_ops.h中定义了GCGE_OPS结构体, 该结构体中包含了需要用户提供的各种操作. 这其中包括一些用户必须提供的基本操作, 以及用户可以选择自己提供或者使用默认操作的向量组相关的操作; 此外GCGE_OPS中还提供了线性求解器的接口, 用户可以选择使用自己提供的线性求解器以及线性求解器结构. 文件gcge_para.h中包含程序运行过程中所需要设置的

各种参数, 用户可以设置其中的参数来对算法运行过程进行控制.

第三层的头文件为gcge_workspace.h, 该文件中定义了GCGE_WORKSPACE的结构, 其中包含了算法执行过程中所需要的工作空间. 基于gcge_ops.h中提供的创建与销毁向量的操作, 以及gcge_para.h中的参数, 用户可以对GCGE_WORKSPACE中的工作空间进行创建或销毁.

依赖第二、三层的头文件, 创建了第四层gcge_rayleighritz.h, gcge_orthogonalize.h, gcge_xpw.h这三个头文件, 其中包含了算法运行过程中具体用到的函数.

依赖于前四层头文件, 创建了第五层的文件gcge_eigsol.h, 其中包含具体实现GCG算法求解特征值问题的主体函数.

第六层包含gcge_solver.h文件, 其中定义了用户可直接使用的GCGE_SOLVER结构体、特征值求解函数的调用接口以及设置用户可调参数的一些函数定义.

2.1.2 SLEPc格式的调用程序结构

SLEPc格式的调用方式如图1 所示. 其中右侧的函数以及中间部分的函数名中不包含SLEPC的都是内核部分的头文件gcge.h中给出的, 中间部分函数名中带有SLEPC的是app部分的头文件gcge_app_slepc.h中给出的接口函数, 用户只需在main函数中调用中间部分的函数就可以进行特征值问题的计算.

具体调用时, 首先读入矩阵, 使用GCGE_SOLVER_SLEPC_Create创建SLEPc格式的求解器GCGE_SOLVER, 设置参数, 然后调用GCGE_SOLVER_SLEPC_Setup函数组装之后, 就可以调用GCGE_SOLVER_Solve函数来进行特征值问题的求解. 求解结束后需要调用GCGE_SOLVER_SLEPC_Free函数释放求解器和所求特征对所占用的内存空间. 另外, 用户可以通过设置打印参数来确定程序运行所需要输出的信息, 如各部分时间统计信息与特征值收敛情况等.

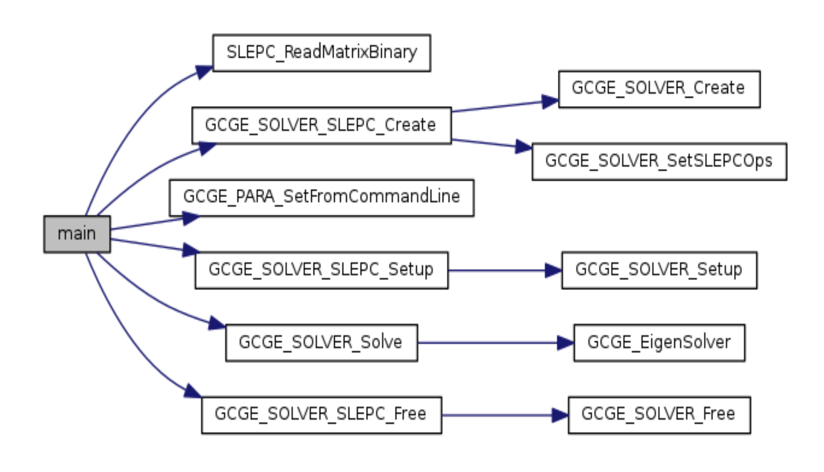


图 1: SLEPc-GCGE调用方式.

3 算法与实现过程的优化

本节基于对GCG算法性质的分析与理解, 介绍在算法实现过程中所采用的一些用来提高算法执行效率的方法, 主要包括高效实现向量组的线性组合、Rayleigh-Ritz问题的求解、正交化过程等的讨论. 为了方便描述和理解, 这里采用MATLAB风格的符号来进行描述.

3.1 充分使用BLAS-3

在算法的执行过程中, 我们充分应用了Level-3的BLAS对计算过程进行加速. 也就是对于串行计算部分和每个进程内部的计算尽量采用矩阵与矩阵相乘的操作. 对于并行部分, 算法中用到的所有分布式存储的向量都以向量组的形式出现. 只要用户的向量组结构支持使用BLAS-3, 就可以在GCGE中直接使用. 比如SLEPc软件包中的BV (BasisVectors)结构, 其局部向量数据是连续存储的, 且相应的向量组操作使用了BLAS-3进行加速, 这个BV结构就可以直接在GCGE软件包中使用. 目前已实现的SLEPc接口就使用了这样的BV结构, 因此在已实现的调用接口中, 对于相同矩阵的特征值进行计算时, SLEPc接口的计算效率是最高的.

另外对于用户只提供单向量局部内积的情况, 我们的多向量内积操作中也进行了统一消息传输的处理, 减少了并行时进程间的通讯时间.

3.2 正交化优化

这一小节介绍软件包中所采用的正交化方法, 并且讨论如何提高向量组正交化的计算效率与并行效率. 也就是说考虑正交化方法的时候需要同时考虑正交化过程的效率与稳定性, 尽量在这两个方面达到一个好的平衡, 以使最后的计算效率达到最优.

除了基本的Modified Gram-Schmidt [15] 正交化方法外, 我们还提供了Modified块正交化方法, Classical块正交化方法, 稳定的块正交化方法, 以及多重正交化方法. 其中Modified块正交化方法是Modified Gram-Schmidt正交化方法的块形式, 两者可以达到相同的精度, 但是块正交化的方式具有更高的计算效率与并行效率. Classical块正交化方法计算量小, 并行效率高, 但不能达到必要的精度而使得特征值求解算法不稳定. 结合以上二者的优势得到了稳定的块正交化方法, 该方法可以得到较高的精度, 且计算效率和并行效率较高. 多重正交化方法几乎可以与Modified Gram-Schmidt正交化方法达到同样的精度, 但多重正交化方法的计算效率和并行效率更高[18].

大部分情况下, 我们推荐使用多重正交化方法(默认方法), 这种方法在第4节的测试中都能达到需要的精度; 如果精度要求很低且想要进一步提高效率, 可以选择使用Classical块正交化方法进行测试; 如果精度要求非常高, 使用默认的正交化方法无法收

敛, 可以使用Modified块正交化方法进行测试. 本小节提供的正交化方法在软件包中都已实现, 可以通过设置相应的正交化参数来进行选择, 以达到用户的稳定性和精度的要求.

3.2.1 Modified块正交化方法

Modified块正交化方法的实现过程描述如下: 给定一个向量组V, V中含有的向量为$V(:,1),\dots,V(:,\texttt{end})$, V中前$\texttt{start}-1$个向量已经是在矩阵B意义下的单位正交基了, 记$V_1=V(:,1:\texttt{start}-1)$, $V_2=V(:,\texttt{start}:\texttt{end})$. 正交化过程是把$V$中后面的向量组$V_2$对前面的向量组$V_1$进行正交, 然后再对$V_2$自己进行单位正交化.

Modified块正交化方法具体的计算过程描述见算法4, 其中reorth_count表示重正交化次数.

Algorithm 4 Modified块正交化方法

计算$V_{\texttt{tmp}}=B\cdot V_1$.
for $\texttt{reorth_count}=1:2$ **do**
  for $i=1:\texttt{start}-1$ **do** ▷ 去掉V_2中V_1方向的分量
    计算$\texttt{prod}=V_{\texttt{tmp}}(:,i)^TV_2$.
    计算$V_2=V_2-V(:,\texttt{i})\texttt{prod}$.
  end for
  for $i=\texttt{start}:\texttt{end}$ **do** ▷ V_2自身做正交化
    计算$\texttt{prod}=V(:,i)^TBV(:,i:\texttt{end})$.
    计算$V(:,i)$的范数$\texttt{norm}=\sqrt{\texttt{prod}(1)}$.
    如果$\texttt{norm}>\tau$, 计算$V(:,i+1:\texttt{end})=V(:,i+1:\texttt{end})-\frac{\texttt{prod}(2:\texttt{end})}{\texttt{norm}}\cdot V(:,i)$.
    否则, 拷贝$V(:,i)=V(:,\texttt{end}-1)$, 同时令$\texttt{end}=\texttt{end}-1$, $i=i-1$.
  end for
end for

正交化算法4需要的临时内存空间为start个向量空间与end-start个浮点数空间. 观察上面的正交化方法可以发现里面的for循环步对每个向量进行处理的时候都需要进行全局通讯, 这是一个不可忽视的时间开销.

Modified块正交化方法本质上是Modified Gram-Schmidt正交化方法的块操作. 在Modified Gram-Schmidt正交化方法中, 每次取一个要进行正交化的向量, 减去在它前面已经正交归一的向量方向上的分量. 为了不损失正交性, 依次减去每个向量方向上的分量. Modified块正交化方法则是同时取所有要进行正交化的向量, 依次减去每个已经正交归一向量方向上的分量. Classical Gram-Schmidt正交化方法相当于对已经正交归一的向量组做一个块处理, 这样可以使计算效率变好, 但往往会造成数值不稳定;

Modified块正交化方法则是反过来对还未完成正交归一的向量组做块处理, 这样既可以提高求解效率, 又能保证数值稳定性.

3.2.2 Classical块正交化

在分布式并行计算环境下, 使用Classical块正交化方法需要进行全局通讯的次数较少, 计算效率高. 但由于数值不稳定的特点(会有正交性损失), 在计算精度要求较高的情况下往往无法使用[15].

记$V_1 = V(:, 1 : \texttt{start} - 1)$, $V_2 = V(:, \texttt{start} : \texttt{end})$. 算法过程如下:

Algorithm 5 `Classical`块正交化方法

1: 计算$V_2 = V_2 - V_1 \cdot (V_1^T B V_2)$. ▷ 去掉$V_2$中$V_1$方向的分量
2: 计算$M = V_2^T B V_2$. ▷ 对V_2自身进行正交化
3: 求解小规模特征值问题$MC = C\Theta$的全部特征对, 且按特征值从小到大排列.
4: 初始化$\texttt{start_}V_2 = 1$, 记$\texttt{length_}V_2 = \texttt{end} - \texttt{start} + 1$.
5: **for** $i = 1 : \texttt{length_}V_2$ **do**
6: 　如果$\Theta_i > \tau$, $C(:, i) = \frac{1}{\sqrt{\Theta_i}} C(:, i)$.
7: 　否则, $\texttt{start_}V_2 = \texttt{start_}V_2 + 1$.
8: **end for**
9: 计算$V_2 = V_2 \cdot C(:, \texttt{start_}V_2 : \texttt{length_}V_2)$.

在Classical块正交化方法中, 去掉V_2中V_1方向分量的方法本质上与Classical Gram-Schmidt正交化方法相同, 会有一定的正交性损失. 我们前面提到Classical Gram-Schmidt正交化方法相当于对已经正交归一的向量组做一个块处理, Classical块正交化方法是既对已经正交归一的向量组V_1做块处理, 也对未完成正交归一的向量组V_2做块处理, 因此并行效率可以进一步得到提高.

3.2.3 稳定的块正交化方法

为了既提高正交化的计算效率, 又能保证稳定性, 我们可以将上面两种方式进行一定的结合来得到下面的正交化方法.

在算法6的第二步中, 对V_2进行正交归一化的时候采用的是与Modified块正交化一样的方式而使得V_2具有很好的正交性. 如果产生V_1的时候也是采用Modified块正交化的方式而导致V_1具有很好的正交性, 这样也会使得在进行第一步(减去V_2在V_1方向上的分量)计算的时候具有很好的数值稳定性. 这就是上面的正交化方法虽然在第一步中采用了块处理的形式依然具有良好的稳定性的原因.

Algorithm 6 稳定的块正交化方法

计算$C = V_1^T B V_2$. ▷ 去掉V_2中V_1方向的分量
计算$V_2 = V_2 - V_1 C$.
计算矩阵Frobenious范数: $\texttt{Inner} = \|C\|_F$.
如果Inner大于正交的阈值, 则重复上面三个步骤的计算直到Inner小于阈值或达到最大重复次数.
for $i = \texttt{start} : \texttt{end}$ **do** ▷ 对V_2自身进行正交化
  计算$\texttt{prod} = V(:,i)^T B V(:,i:\texttt{end})$.
  计算$V(:,i)$的范数$\texttt{norm} = \sqrt{\texttt{prod}(1)}$.
  如果$\texttt{norm} > \tau$, 计算$V(:,i+1:\texttt{end}) = V(:,i+1:\texttt{end}) - \frac{\texttt{prod}(2:\texttt{end})}{\texttt{norm}} \cdot V(:,i)$;
  否则, 令$V(:,i) = V(:,\texttt{end})$, 同时令$\texttt{end} = \texttt{end} - 1,\ i = i - 1$.
end for

3.2.4 多重正交化

在正交化优化过程中可以发现尽量对已收敛的向量组进行统一正交化(块的形式)可以减少进程间消息发送的次数, 同时可以将部分BLAS-2的操作转化为BLAS-3的操作. 因此我们考虑使用二分递归的方式来尽量对已收敛的向量进行统一的正交化处理. 这种方法可以将几乎全部的BLAS-2的操作转化为BLAS-3的操作, 且进程间消息传输的次数nc与向量组中的向量个数nv的关系为: $\texttt{nc} = \mathcal{O}(\texttt{nv})$. 由于这里设计正交化方法的思想来源于多重网格迭代算法的启发, 所以我们将这种方法称为多重正交化方法. 后来我们在文献综述时发现文章[18]也讨论了类似的算法.

考虑对向量组V中的第start位置到end位置的向量进行正交化, 算法过程见算法7, 其中size_V表示V中的向量个数, $\lceil \texttt{length}/2 \rceil$表示不小于length/2的最小的整数.

算法7充分利用了稳定的块正交化方法的优点, 尽量减少要进行自身正交化向量组的个数(只对单个向量进行正交归一化). 这样的正交化方式除了对单个向量进行归一化外其它的部分都可以用BLAS-3进行计算, 从而可以充分提高计算效率和并行效率.

3.3 减少计算子空间特征值的个数与向量组线性组合的次数

本小节将目光转向如何高效地计算Rayleigh-Ritz问题以得到新的特征向量逼近. 这个过程包括串行求解一个小规模的特征值问题, 应该尽量减少其在整个计算时间中所占的比重. 求解Rayleigh-Ritz问题的计算过程包括: 计算小规模矩阵$\bar{A} = V^T A V$, 然后求解标准特征值问题$\bar{A} C = C\Lambda$, 最后将计算出来的特征向量C返回到长向量组V进行向量的线性组合以得到新的特征向量逼近X与向量组P.

Algorithm 7 多重正交化Multi_Orth(V, start, end)

```
计算size_V = end − start + 1.
令mid = start + ⌈size_V/2⌉ − 1, old_mid = mid.
if size_V == 1 then
    计算norm = ‖V(:, start)‖_2.
    if norm > τ then
        计算V(:, start) = V(:, start)/norm.
    else
        令end = start.
    end if
else
    调用Multi_Orth(V, start, mid), 返回V和mid.
    if mid < old_mid + 1 then
        记n_zero = old_mid − mid, 更新end = end − n_zero.
        拷贝V(:, mid : old_mid) = V(:, end : end + n_zero).
    end if
    记V_1 = V(:, start : mid),V_2 = V(:, mid + 1 : end).
    重复计算V_2 = V_2 − V_1(V_1^T · V_2)直到达到正交化的精度要求, 最多计算3次.
    调用Multi_Orth(V, mid + 1, end).
end if
返回V和end.
```


3.3.1 计算小规模矩阵

在求解Rayleigh-Ritz问题的时候, 我们需要先把相应的小规模矩阵V^TAV和V^TBV计算出来. 由于在对V进行正交化的时候采用的是矩阵B的内积, V^TBV就是一个单位矩阵, 所以不需要计算. 那么计算小规模矩阵的过程主要是形成矩阵$\bar{A} = V^TAV$.

我们知道V的形式为$V = [X, P, W]$. 为了与前面算法的描述相对应, 把X表示成两部分$X = [X_0, X_1]$, 其中X_0表示已收敛的特征向量组. 本小节中我们均假设已有ℓ个特征对收敛($0 \le \ell < \texttt{nev}$), X_1表示未收敛的近似特征向量组. 由算法的过程可以知道, 这里的X是求解上一次迭代的Rayleigh-Ritz问题得到的, 即$X = VC$, 其中C满足上一次迭代中的Rayleigh-Ritz问题:

$$\bar{A}_{\texttt{old}}C = C\Lambda, \tag{1}$$

其中C是一个正交矩阵, Λ是一个对角矩阵, $\bar{A}_{\texttt{old}}$是已知的上一次迭代的小规模矩阵. 矩阵C中的每一列都是矩阵$\bar{A}_{\texttt{old}}$的一个特征向量, 其相应的特征值为对角阵Λ相应位置的对

角元素. 由此可以知道矩阵$\bar{A}_{\tt old}$有如下的谱分解:

$$\bar{A}_{\tt old} = C\Lambda C^T. \tag{2}$$

我们可以把特征向量组C分解成两部分$C = [C_x, C_x^{\perp}]$, 其中C_x与向量组X相对应, 即$X = VC_x$. 与X的结构$X = [X_0, X_1]$相对应, 我们也把C_x分解成$C_x = [C_0, C_1]$.

为了继续进行GCG迭代, 需要形成下一次迭代Rayleigh-Ritz问题的矩阵$\bar{A}_{\tt new}$如下:

$$\bar{A}_{\tt new} = \begin{pmatrix} X^TAX & X^TAP & X^TAW \\ P^TAX & P^TAP & P^TAW \\ W^TAX & W^TAP & W^TAW \end{pmatrix}. \tag{3}$$

对于X^TAX部分的矩阵块有如下的性质

$$X^TAX = C_x^TV^TAVC_x = C_x^T\bar{A}_{\tt old}C_x = C_x^TC\Lambda C^TC_x = \Lambda_x, \tag{4}$$

其中Λ_x表示在对角阵Λ中与向量组X相对应的对角元素. 由此可以知道我们并不需要去直接计算矩阵元素X^TAX, 而只需把上一次迭代中的对角阵Λ_x放入$\bar{A}_{\tt new}$中即可.

下面来考虑如何计算矩阵X^TAP和P^TAP. 为此我们先来考虑如何生成向量组P. 由GCG算法的定义可以知道$P = VC_p$, 其中C_p表示与向量组P相对应的系数矩阵C_p. 为了得到系数矩阵C_p, 我们把向量组C_1分解成如下的形式:

$$C_1 = \begin{bmatrix} C_{1,x} \\ C_{1,p} \end{bmatrix}.$$

如此首先可以设置向量组$\widetilde{C}_p$ 如下:

$$\widetilde{C}_p = \begin{bmatrix} 0 \\ C_{1,p} \end{bmatrix}.$$

接下来对小规模向量组$[C_0, C_1, \widetilde{C}_p]$进行$L^2$正交化. 注意这里的$[C_0, C_1]$已经是单位正交基, 所以只需要把$\widetilde{C}_p$对前面的$[C_0, C_1]$进行正交化得到$\bar{C}_p$, 然后再对$\bar{C}_p$自身进行正交化即可. 设经过正交化之后得到的向量组记为$[C_0, C_1, C_p]$. 注意这里的正交化是小规模向量组的正交化, 并不需要对长向量组$[X, P]$直接进行正交化.

由上面的讨论可以得到下面计算X^TAP和P^TAP的方式:

$$\begin{aligned} X^TAP &= C_x^TV^TAVC_p = C_x^T\bar{A}_{\tt old}C_p = C_x^TC\Lambda C^TC_p = C_x^T[C_x, C_x^{\perp}]\Lambda[C_x, C_x^{\perp}]^TC_p \\ &= C_x^T(C_x\Lambda_xC_x^T + C_x^{\perp}\Lambda_x^{\perp}(C_x^{\perp})^T)C_p = C_x^TC_x\Lambda_xC_x^TC_p = 0, \end{aligned} \tag{5}$$

和

$$P^TAP = C_p^T\bar{A}_{\tt old}C_p. \tag{6}$$

最后由(3), (4), (5)和(6) 可以得到矩阵$\bar{A}_{\texttt{new}}$的结构如下

$$\bar{A}_{\texttt{new}} = \begin{pmatrix} D_1 & 0 & a_1 \\ 0 & \alpha_1 & a_2 \\ a_1^T & a_2^T & \alpha_2 \end{pmatrix}, \tag{7}$$

其中$\alpha_1 = C_p^T \bar{A}_{\texttt{old}} C_p$, $\alpha_2 = W^T AW$, $a_1 = X^T AW$和$a_2 = P^T AW$. 也就是说这里的a_1, a_2和α_2是需要真正针对大规模向量组进行计算的, D_1直接就是对角矩阵Λ中与向量组X相对应的部分Λ_x, α_1可以通过小规模矩阵的计算$C_p^T \bar{A}_{\texttt{old}} C_p$得到.

因为X_0已经达到收敛要求, 我们可以近似地认为$AX_0 = BX_0\Lambda_0$, 于是近似地有$X_0^T AW = \Lambda_0 X_0^T BW = 0$. 那么(7) 近似地有下面的形式

$$\bar{A}_{\texttt{new}} = \begin{pmatrix} \Lambda_0 & 0 & 0 & 0 \\ 0 & \Lambda_1 & 0 & \bar{a}_1 \\ 0 & 0 & \alpha_1 & a_2 \\ 0 & \bar{a}_1^T & a_2^T & \alpha_2 \end{pmatrix}, \tag{8}$$

其中Λ_0表示已收敛的特征值构成的对角矩阵, Λ_1表示未收敛的特征值构成的对角矩阵, $\bar{a}_1 = X_1^T AW$. 由(8) 可知, 矩阵$\bar{A}_{\texttt{new}}$是可约的. 因此只需对矩阵

$$\bar{A}_{\texttt{sub}} = \begin{pmatrix} \Lambda_1 & 0 & \bar{a}_1 \\ 0 & \alpha_1 & a_2 \\ \bar{a}_1^T & a_2^T & \alpha_2 \end{pmatrix}, \tag{9}$$

的特征对进行求解, 即求解$\bar{A}_{\texttt{sub}}\widetilde{C} = \widetilde{C}\Lambda$, 那么$\bar{A}_{\texttt{new}}$的相应的特征向量为

$$\begin{pmatrix} I_\ell & 0 \\ 0 & \widetilde{C} \end{pmatrix}. \tag{10}$$

本小节介绍的方法与[5]中的方式基本类似, 但是计算α_1的部分不一样. 这样做的原因是我们可以不用计算矩阵$\bar{A}_{\texttt{new}}$的所有特征对, 而只需计算C_x部分即可. 且使用优化(8)时可以使实际要求解特征值问题的矩阵规模更小, 这样就为下一步减少计算小规模特征值问题$\bar{A}_{\texttt{new}}C = C\Lambda$的时间提供了条件.

3.3.2 使用dsyevx代替dsyev来求解部分特征值

在上一小节计算出Rayleigh-Ritz特征值问题的矩阵$\bar{A}_{\texttt{new}}$之后, 接下来就是要求解特征值问题$\bar{A}_{\texttt{new}}C = C\Lambda$. 最常用的方式就是调用LAPACK [1]中的函数dsyev来求解所有特征对. 为了尽量减少计算量, 我们只求解子空间矩阵$\bar{A}_{\texttt{new}}$的第$\ell+1$到第`nev`个特征值和相应的特征向量C_x, 具体是调用dsyevx来进行计算. 当然在下一小节的讨论中, 我们会把这一部分的计算量进一步地减少.

3.3.3 进一步减少计算子空间特征值的个数与向量组线性组合的次数

在计算Rayleigh-Ritz问题的过程中, 我们需要用串行的方式求解如下特征值问题:

$$\bar{A}_{\mathtt{new}} C = C\Lambda. \tag{11}$$

在具体的迭代过程中, 不同的特征对是依次收敛的. 由于V中会保存已收敛的特征向量, 所以对于特征值问题(11), 相应的单位向量$(0, ..., 0, 1, 0, ...0)$就是其相应于已收敛特征对的一个特征向量, 其中1所在的位置即为这个已收敛特征对的编号. 也就是说在求解特征值问题(11)的时候, 对已经收敛的特征向量不需要再进行求解了, 而只需要求解还未收敛的那些特征向量, 即与X_1相对应的特征对. 这样就减少了调用dsyevx计算特征对的个数而进一步减少计算时间.

更进一步, 在已经有特征向量收敛的情况下, 当进行小规模向量组正交化(计算$[C_x, C_p]$)的时候可以进一步减少计算量. 设已有ℓ个特征对收敛, 即特征值问题(11)的前ℓ个特征向量为$[I_\ell, 0]^T$, 所以只需要将新得到的子空间向量组$[C_x, C_p]$每列的前ℓ个元素赋为0, 就可使其与前ℓ个特征向量正交. 然后再对这些子空间特征向量组的非零部分进行正交化就可以得到正交向量组$[C_x, C_p]$.

为了方便理解, 我们通过下面的具体过程来阐述上面的方法. 首先通过算法的构造过程可以知道矩阵$\bar{A}_{\mathtt{new}}$具有如下的结构:

$$\bar{A}_{\mathtt{new}} = \begin{pmatrix} D_1 & 0 & a_1 \\ 0 & \alpha_1 & a_2 \\ a_1^T & a_2^T & \alpha_2 \end{pmatrix}.$$

在迭代过程中, 对于第i个已收敛的特征对, 其特征向量已达到收敛准则, 可以不必更新, 因此可将其子空间特征向量记为e_i. 设已有ℓ个连续的特征对收敛, 那只需要再求解$\bar{A}_{\mathtt{new}}$的第$\ell+1$到`dimX`个特征对. 这里`dimX`表示向量组$[X_0, X_1]$中的向量个数. 那么可以知道$\bar{A}_{\mathtt{new}}$的前`dimX`个特征向量如下

$$C_x = \begin{pmatrix} I_\ell & C_{12} \\ 0 & C_{22} \end{pmatrix}.$$

要使X的各列均正交, 首先将C_{12}设为0, 再对C_{22} 进行正交化以得到新的向量组$\bar{C}_{22}$. 记正交化后的特征向量为

$$\bar{C}_x = \begin{pmatrix} I_\ell & 0 \\ 0 & \bar{C}_{22} \end{pmatrix}.$$

因为$\ell < \texttt{dimX}$, 所以$\bar{C}_x$可以写成下面的形式

$$\bar{C}_x = \begin{pmatrix} I_\ell & 0 \\ 0 & \bar{C}_{xx} \\ 0 & \bar{C}_{xp} \\ 0 & \bar{C}_{xw} \end{pmatrix},$$

其中$\bar{C}_{xx}$, $\bar{C}_{xp}$和$\bar{C}_{xw}$分别与向量组V中的X, P和W部分相对应. 这样就可以构造出如下的与长向量组$[X, P]$所对应的系数向量组

$$[\bar{C}_x, C_p] = \begin{pmatrix} I_\ell & 0 & 0 \\ 0 & \bar{C}_{xx} & 0 \\ 0 & \bar{C}_{xp} & \bar{C}_{pp} \\ 0 & \bar{C}_{xw} & \bar{C}_{pw} \end{pmatrix}.$$

由此可以知道要对P向量组部分进行正交化, 只需要对下面的子块进行正交化即可

$$\begin{pmatrix} \bar{C}_{xx} & 0 \\ \bar{C}_{xp} & \bar{C}_{pp} \\ \bar{C}_{xw} & \bar{C}_{pw} \end{pmatrix}.$$

对上面的向量组进行正交化之后得到如下的向量组

$$[\bar{C}_x \ \bar{C}_p] = \begin{pmatrix} I_\ell & 0 & 0 \\ 0 & \bar{C}_{xx} & \bar{C}_{px} \\ 0 & \bar{C}_{xp} & \bar{C}_{pp} \\ 0 & \bar{C}_{xw} & \bar{C}_{pw} \end{pmatrix}.$$

然后利用存储在V中的基向量组进行线性组合以得到新的向量组$[X, P] = V[\bar{C}_x, \bar{C}_p]$. 根据上面给出的系数矩阵$[\bar{C}_x, \bar{C}_p]$, 对于已经收敛的特征方向$X(:, 1 : \ell)$ 不需要进行更新, 并且$X(:, \ell + 1 : \texttt{dimX})$的更新采用如下方式

$$X(:, \ell + 1 : \texttt{dimX}) = V(:, \ell + 1 : \texttt{dimXPW}) \cdot \begin{pmatrix} \bar{C}_{xx} \\ \bar{C}_{xp} \\ \bar{C}_{xw} \end{pmatrix},$$

向量组P的更新方式如下

$$P = V(:, \ell + 1 : \texttt{dimXPW}) \cdot \begin{pmatrix} \bar{C}_{px} \\ \bar{C}_{pp} \\ \bar{C}_{pw} \end{pmatrix},$$

其中dimXPW表示基向量组$[X, P, W]$中的向量个数.

如果一个已收敛的特征值是重的, 且它的特征空间还未完全收敛, 那么需要对重特征值进行一定的处理. 在检查未收敛特征对的收敛性时, 我们同时对相应特征值的重数进行检查. 如果这个特征值与前一个特征值的相对误差小于一个阈值(这里默认的阈值1e-6), 那就认为这个特征值与前一个特征值是重的. 如果当前检查的特征对已收敛, 那么就检查其与前一个特征值是否是重特征值, 如果是重的且前一个特征对被标记为未收敛, 那么就将当前的特征对也标记为未收敛; 如果当前检查的特征对未收敛, 也检查其与前一个特征值是否是重特征值, 如果是重的且前一个特征对被标记为已收敛, 那么就将前一个特征对也标记为未收敛. 检查收敛性直到连续未收敛的特征对个数达到一定的上限(默认为10), 之后的特征对都标记为未收敛, 或者到最后一个特征对为止. 记从第1个特征对开始连续收敛的最后一个特征对的编号为$\ell_{\texttt{min}}$, 则求解子空间矩阵特征值问题(11)时要从第$\ell_{\texttt{min}}+1$个开始计算, P与W向量组的构造是相应于X中标记为未收敛的向量进行.

3.4 计算子空间特征值问题的优化

3.4.1 只让0号进程计算子空间特征值问题

由于各进程计算的是相同子空间上的Rayleigh-Ritz特征值问题, 且计算完毕后需要由0号进程把相应的特征对广播到各个进程以保持所有进程的数据统一, 在广播前可能会有进程等待的情况, 所以只由0号进程进行计算, 可以在一定程度上减少这部分的计算时间.

3.4.2 每个进程计算一部分特征值, 再统一到所有进程

当求解特征值个数较多时, 由于计算子空间特征值问题的时间与求解特征值的个数是超线性关系, 这部分的计算时间占比会变得很高. 为了减少计算时间, 我们将特征值问题(11)平均分配到所有进程进行求解, 每个进程只计算少量的几个特征对, 这样就可以大大地减少求解子空间特征值问题(11)的计算时间. 计算结束后, 使用MPI_Allgatherv将所有进程的特征对信息进行统一. 在进程数不是那么多的时候, 计算特征对的时间相比MPI_Allgatherv的时间要多很多, 此时采用这样的方式就可以提高计算效率. 为了使所有进程的子空间特征对信息统一, 我们需要对它们进行广播. 这部分的消息传输是必不可少的, 当进程数很多的时候, 可以考虑只让其中一部分的进程进行计算从而减少消息传递所占的时间比重. 为了避免一些进程中计算的特征值个数太少又需要其对所有进程发送它所计算的特征值, 我们设置让每个进程至少计算一定个数的特征值, 默认每个进程至少计算10个.

3.5 使用MKL进行优化

为了进一步减少计算时间, 当使用Intel编译器时可以调用mkl的BLAS与LAPACK库. 一般地, 如果计算设备上用的是Intel的CPU并且同时有mkl的BLAS和LAPACK库的时候, 建议使用它们.

4 测试结果

以下进行的所有测试的计算环境为中国科学院数学与系统科学研究院科学与工程计算国家重点实验室LSSC4 计算集群, 提交任务时采用的是big队列. 每个节点包括2颗主频为2.3GHz的Intel Xeon Gold 6140 18核Purley处理器和192GB内存, 更详细的介绍可以参看http://lsec.cc.ac.cn/chinese/lsec/LSSC-IVintroduction.pdf.

我们对以下四组对称矩阵进行了测试. 其中前3个矩阵取自Suite Sparse Matrix

表 1: 测试矩阵.

编号	矩阵	维数	非零元个数
1	Andrews	60000	760154
2	Ga3As3H12	61349	5970947
3	Ga10As10H30	113081	6115633
4	模态分析矩阵	2736987	189967149

Collection[1], Andrews矩阵的第一个特征值接近于0, Ga3As3H12与Ga10As10H30矩阵均有大量负特征值, 这三个矩阵的特征值分布均非常稠密; 模态分析矩阵是从某建筑体的模态分析中导出的广义代数特征值问题矩阵[6, 7, 19].

在软件包GCGE所提供的接口中, 进行了向量组计算优化的SLEPc接口是计算效率最高的, 因此以下的数值计算测试均使用SLEPc的调用接口执行. 为了进行对比, 我们还使用SLEPc中的LOBPCG以及Jacobi-Davidson解法器进行了测试.

4.1 优化策略对计算效率提升的测试

在这一小节, 我们用实际的数值算例来测试本文中介绍的算法优化的效果. 这里求解的特征值问题中的矩阵为Andrews矩阵, 收敛准则为1e-4, 在单个节点上用24个进程求解前600个最小端的特征值和相应的特征向量. 表2展示了使用文中优化过程所带来时间减少的情况, 其中RR过程指Rayleigh-Ritz过程的计算时间.

[1]详见https://sparse.tamu.edu

分批计算(120)表示进行分批计算时设置参数block_size为120, 根据经验, 我们设置分批计算时参数block_size的默认值为nev/5. 这种方式会减少每次迭代中向量组P与向量组W的计算量, 同时也可以减小Rayleigh-Ritz过程的计算规模, 从计算结果可以看到, 计算P, 计算W以及Rayleigh-Ritz过程的计算时间均有减少.

向量运算优化表示对算法中向量的操作尽量使用向量组统一计算的方式, 这样可以尽可能地使用BLAS-3对算法进行加速. 从计算结果可以看到, 算法各部分的计算时间均有明显减少, 尤其是计算X与P的部分, 这是因为这两部分的计算量主要是进行可以完全转化为BLAS-3操作的向量组线性组合的操作.

在正交化优化中, W的正交化采用的是不稳定但最高效的Classical块正交化方法, 即算法5, P的正交化采用的是稳定且高效的多重正交化方法, 即算法7. 因此计算P 和W两部分的计算时间有明显地减少.

最后的稠密特征值优化, 是在Rayleigh-Ritz过程中使用不同进程分别求解一部分特征对, 再统一进行MPI的全收集操作. 这使得Rayleigh-Ritz过程的计算时间有明显地减少.

表 2: Andrews矩阵, 收敛准则取1e-4, 使用不同优化方法的时间对比(秒).

优化方式	计算X	计算P	计算W	RR过程	总时间
原始GCG	25.06	32.98	101.50	33.42	193.49
分批计算(120)	23.29	8.49	52.97	15.43	101.77
向量运算优化	2.51	2.05	50.81	10.19	65.98
正交化优化	1.42	0.84	16.74	9.79	29.20
稠密矩阵特征值优化	1.43	0.83	16.94	4.13	23.82

4.2 求解特征值个数与计算时间关系的测试

为了测试GCG算法求解特征值问题的时间与所求特征值个数的关系, 我们使用一个节点上的24个进程对表1中的前3个矩阵进行了测试. 分别计算了75-1200个特征值, 测试中均使用了绝对残差($\|Ax-\lambda Bx\|_2/\|x\|_2$)作为收敛准则, 分别对比了收敛准则取1e-4和1e-12的结果. 这里测试使用GCG算法时, 分批计算的参数均取block_size = nev/5, 且求解线性方程组时均使用10次CG迭代, 使用的正交化方法均为多重正交化方法(这种正交化方法在精度、稳定性和可扩展性方面达到了很好的平衡). 由于LOBPCG方法所展现出来的良好的计算效率与可扩展性, 本小节与4.3小节的计算结果均与SLEPc中的LOBPCG进行了比较.

4.2.1 Andrews矩阵

对Andrews矩阵, 收敛准则取1e-4时. 经过实际的数值测试, 我们发现LOBPCG的参数−`eps_lobpcg_blocksize`设置为`nev/6`, −`eps_lobpcg_restart`设置为0.1的时候计算效率最高. 我们采用这样的参数设置所得到的计算结果来进行比较. 且在此之后的参数−`eps_lobpcg_restart`均设置为0.1. 两种方法的计算时间对比情况见表3与图2. 从计算结果的对比可以发现GCGE相比LOBPCG, 计算效率有约3-4倍的提升.

表 3: Andrews矩阵, 收敛准则取1e-4时两种方法的计算时间对比(秒).

nev	GCGE	LOBPCG
75	1.98	10.12
150	4.51	24.42
300	10.59	60.95
600	28.20	146.62
1200	103.56	405.38

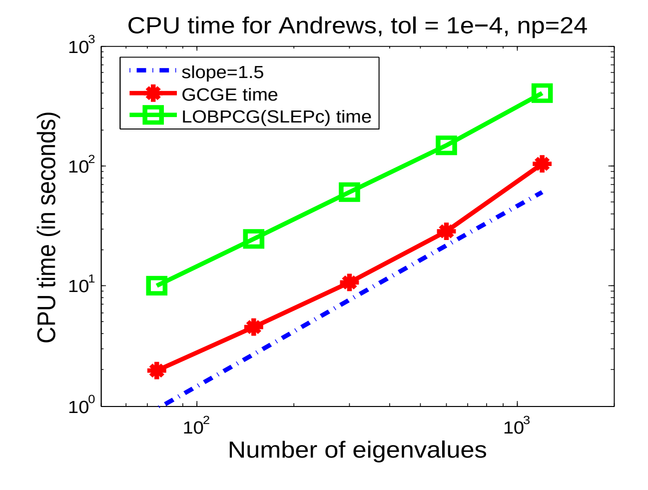


图 2: Andrews矩阵, 收敛准则取1e-4时两种方法的计算时间对比.

收敛准则取1e-12 时, LOBPCG如果仍使用收敛准则为1e-4时的最优参数, 在特征值个数分别为150,600,1200 时计算失败, 并且特征值个数分别为75,300时, 将参数−`eps_lobpcg_blocksize`设置为`nev`或`nev/6`的时候计算时间几乎一致. 因此将参数−`eps_lobpcg_blocksize`设置为`nev`来对LOBPCG进行测试. 两种方法计算时间的对比情况见表4与图3. 由其中的数值结果可以发现, GCGE计算更稳定, 且计算效率相对于LOBPCG提升约6倍.

表 4: Andrews矩阵, 收敛准则取1e-12时两种方法的计算时间对比(秒).

nev	GCGE	LOBPCG
75	5.46	24.20
150	11.54	62.75
300	28.33	168.08
600	70.63	512.89
1200	249.04	1931.54

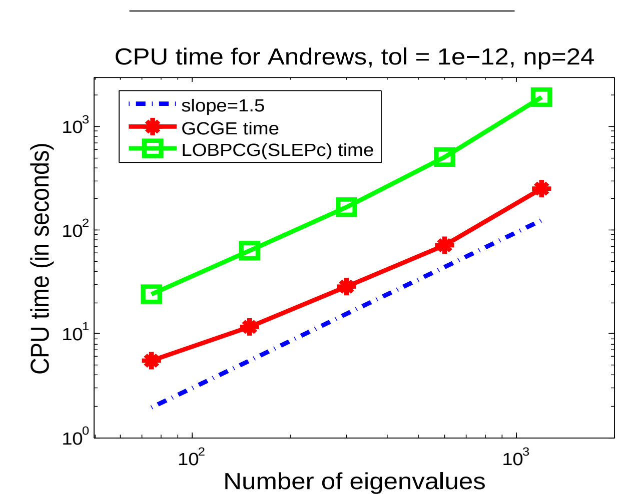


图 3: Andrews矩阵, 收敛准则取1e-12时两种方法的计算时间对比.

4.2.2 Ga3As3H12矩阵

本小节对Ga3As3H12矩阵进行测试. 当收敛准则为τ = 1e-4时, LOBPCG的参数-eps_lobpcg_blocksize设置为nev/6, 相应的计算时间对比情况见表5与图4. 由其中的结果我们可以发现GCGE的计算效率比LOBPCG有约2-3倍的提升.

表 5: Ga3As3H12矩阵, 收敛准则取1e-4时两种方法的计算时间对比(秒).

nev	GCGE	LOBPCG
75	8.36	20.73
150	14.91	41.88
300	27.80	84.33
600	57.52	160.32
1200	148.84	378.99

当收敛准则为τ = 1e-12时, 使用LOBPCG进行计算的时候往往失败或者在一定时间内无法收敛, 计算结果对比情况见表6, 其中LOBPCG的参数-eps_lobpcg_blocksize

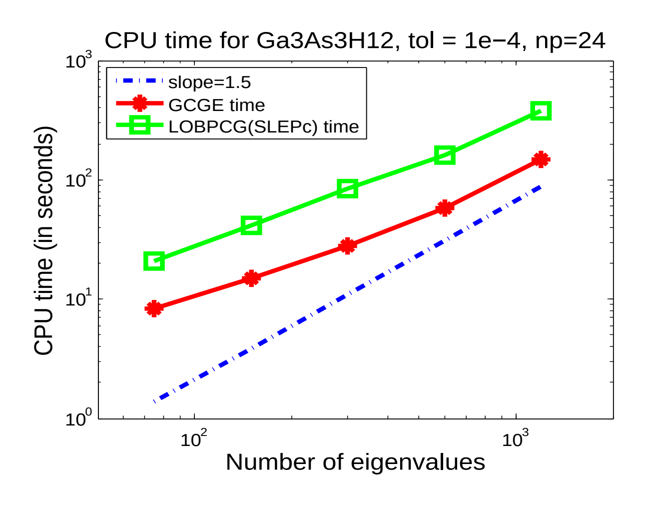


图 4: Ga3As3H12矩阵, 收敛准则取1e-4时两种方法的计算时间对比.

设置为nev. 此时可以发现GCGE有明显的稳定性和效率的优势.

表 6: Ga3As3H12矩阵, 收敛准则取1e-12时两种方法的计算时间对比(秒).

nev	GCGE	LOBPCG
75	23.45	30000步收敛了19个
150	40.78	Error in Lapack DSYGVD 127
300	75.67	运行超过24小时候未收敛
600	149.93	运行超过24小时候未收敛
1200	368.65	运行超过24小时候未收敛

4.2.3 Ga10As10H30矩阵

本小节对Ga10As10H30矩阵进行测试, 当收敛准则$\tau = 1e\text{-}4$时, LOBPCG的参数−eps_lobpcg_blocksize设置为nev, 两种方法的计算时间对比情况见表7与图5. 由其中的数值结果可以发现GCGE的计算效率相对于LOBPCG有约2-3倍的提高.

表 7: Ga10As10H30矩阵, 收敛准则取1e-4时两种方法的计算时间对比(秒).

nev	GCGE	LOBPCG
75	9.37	24.18
150	20.07	52.36
300	40.32	109.53
600	78.72	231.73
1200	205.24	559.18

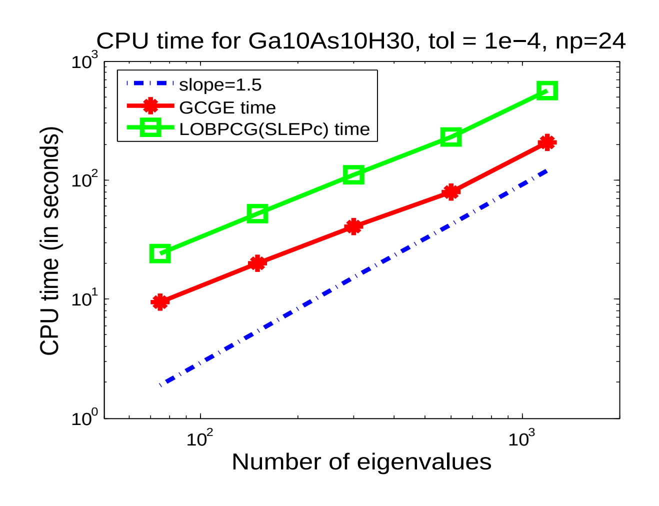


图 5: Ga10As10H30矩阵, 收敛准则取1e-4时两种方法的计算时间对比.

当收敛准则为$\tau = 1\text{e-}12$时, 使用LOBPCG进行求解往往计算失败或者在一定时间内无法收敛, 计算结果的对比情况见表8, 其中LOBPCG的结果是参数-eps_lobpcg_blocksize设置为nev的时候得到的. 同样可以发现GCGE有明显的稳定性和效率上的优势.

表 8: Ga10As10H30矩阵, 收敛准则取1e-12时两种方法的计算时间对比(秒).

nev	GCGE	LOBPCG
75	25.17	Error in Lapack DSYGVD 78
150	55.16	Error in Lapack DSYGVD 134
300	110.07	运行超过24小时未收敛
600	212.47	运行超过24小时未收敛
1200	526.26	运行超过24小时未收敛

4.3 强可拓展性测试

为了进行算法强可拓展性的测试, 我们对Andrews矩阵, Ga10As10H30矩阵, 分别使用1-32个进程进行测试. 这里同样将GCGE与SLEPc中的LOBPCG进行对比, 具体的计算结果见图6与图7. 对比图6和7可以发现, GCGE有与LOBPCG几乎一样的强可拓展性, 并且GCGE具有更好的计算效率.

4.4 大规模计算时算法的稳定性与计算效率测试

这一小节考虑求解某种复杂建筑体模态分析所导致的特征值问题. 模态分析问题的

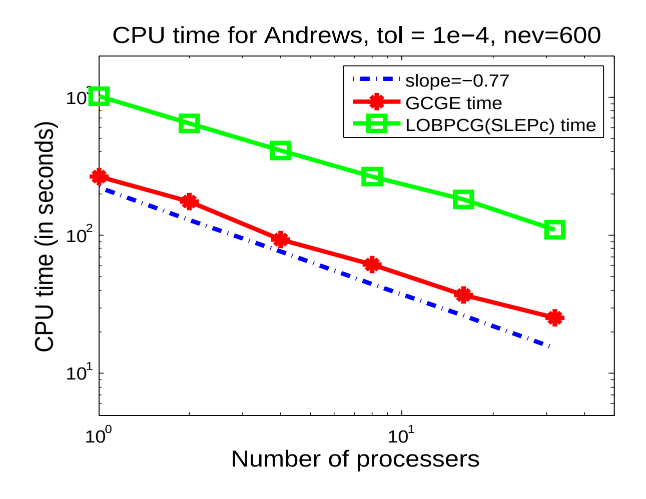


图 6: Andrews矩阵, 收敛准则取1e-4, 与LOBPCG强可拓展性对比.

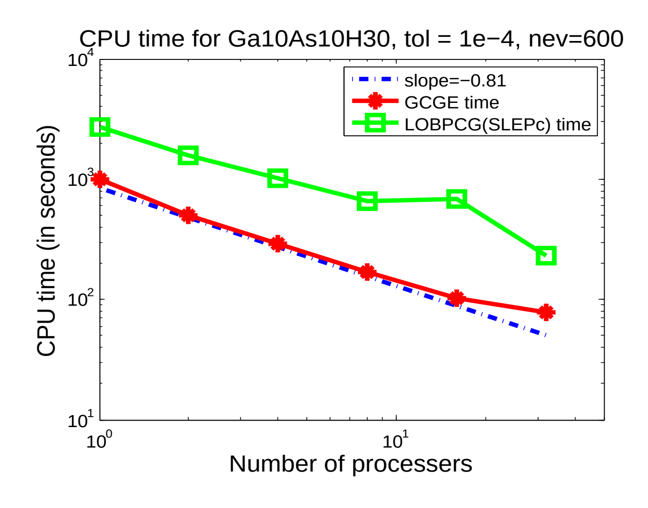


图 7: Ga10As10H30矩阵, 收敛准则取1e-4, 与LOBPCG强可拓展性对比.

力学模型如下

$$\left\{\begin{array}{rcl} -G\mathbf{\Delta u}-(\mu+G)\nabla(\nabla\cdot\mathbf{u}) & = & \lambda\mathbf{u}, \quad \text{in } \Omega, \\ \mathbf{u} & = & 0, \quad \text{on } \partial\Omega, \end{array}\right. \tag{12}$$

其中

$$\mathbf{\Delta}=\begin{bmatrix}\Delta & & \\ & \Delta & \\ & & \Delta\end{bmatrix}, \ \nabla=\begin{bmatrix}\frac{\partial}{\partial x}\\ \frac{\partial}{\partial y}\\ \frac{\partial}{\partial z}\end{bmatrix}, \ \nabla\cdot=\left[\frac{\partial}{\partial x},\frac{\partial}{\partial y},\frac{\partial}{\partial z}\right], \ \mathbf{u}=\begin{bmatrix}u_x\\ u_y\\ u_z\end{bmatrix}.$$

这里G为材料剪切模量(剪切弹性模量), μ为材料体积模量, 二者统称为拉梅常数(Lamé constant), 其表达式如下:

$$G=\frac{E}{2(1+\nu)}, \ \mu=\frac{E\nu}{(1+\nu)(1-2\nu)},$$

其中E为弹性体的杨氏模量, ν为Poisson比. 由于弹性体可能由多种材料构成, 所以拉梅常数在区域的不同位置可能取不同的值. 本小节考虑的力学模型有如下的应用特征

表 9: 模态分析问题的部分应用特征.

模型	Poisson 比	杨氏模量
某建筑体模态分析问题	0.15, 0.3	7.0e7, 2.48e7, 2.1e8

本小节所测试的矩阵是使用表9中的参数, 并用有限元并行计算框架Panda [7, 19] 进行离散得到的. 这组矩阵是对称正定的, 但由于使用了多种材料, 且网格剖分的精细程度不同, 导致该组矩阵具有很强的多尺度特性. 这就使得矩阵条件数很大, 对解法器的稳定性要求很高, 给特征值的计算带来了很大的挑战. 表10中给出了这组矩阵的部分数学性质, 其中A, B分别表示该广义特征值问题的刚度矩阵与质量矩阵. 由表10可以看出这组矩阵具有很强的多尺度特性且条件数很大.

表 10: 模态分析问题所产生矩阵的性质.

性质	A	B
行数	273,6987	273,6987
非零元个数	1,8996,7149	1,8996,7149
稀疏度	69.41	69.41
对角线元素最小值	9.90e+08	3.88e-02
对角线元素最大值	3.45e+11	4.29e+01
非对角线元素最小值	-1.67e+11	2.34e-03
非对角线元素最大值	1.69e+11	1.60e+01
非对角线元素最小绝对值	2.91e-11	2.34e-03
非对角线元素最大绝对值	1.69e+11	1.60e+01
最小对角占有率	1.32e-02	2.60e-02
对称性	sym (1.0e+00)	sym (1.0e-10)
条件数	8.57e+07	7.74e+03

我们使用SLEPc软件包中的几种常用的算法对本小节的问题进行了测试. 使用Krylov-Schur方法求解时, 可以达到要求的计算精度, 且收敛速度快. 缺点是该方法需要精确求解迭代过程中的线性方程组, 由于本小节的矩阵的条件数较大, 只能使用LU分解才能求解, 因此并行可拓展性较差. LOBPCG方法不需要精确求解线性方程组, 可以

有较好的并行可拓展性, 但该方法稳定性较差, 求解本小节的矩阵特征值问题时总是求解失败. Jacobi-Davidson方法不需要精确求解线性方程组, 并行可扩展性较好, 但该方法较为复杂, 算法参数很多, 难以为本小节的矩阵找到最优的求解参数以及构造良好的预条件矩阵.

本小节使用2736987阶的矩阵[6, 7]进行测试, 收敛准则为相对残差$\|Ax-\lambda Bx\|_2/(|\lambda|\cdot\|x\|_2) < 5\text{e-}2$, 使用72到576个进程, 分别求解最小端的100和1000个特征值. 根据上面的分析, 这里我们将GCGE与SLEPc中的Jacobi-Davidson解法器进行对比. 使用GCGE求解时, 用500次CG迭代计算W. 经过多次试验, 也为Jacobi-Davidson方法选取了已知最好的参数. 相应的对比结果见图8. 从图中可以发现GCGE与Jacobi-Davidson均有较好的强可拓展性, 且GCGE的计算速度相比Jacobi-Davidson方法有约3-5倍的提高.

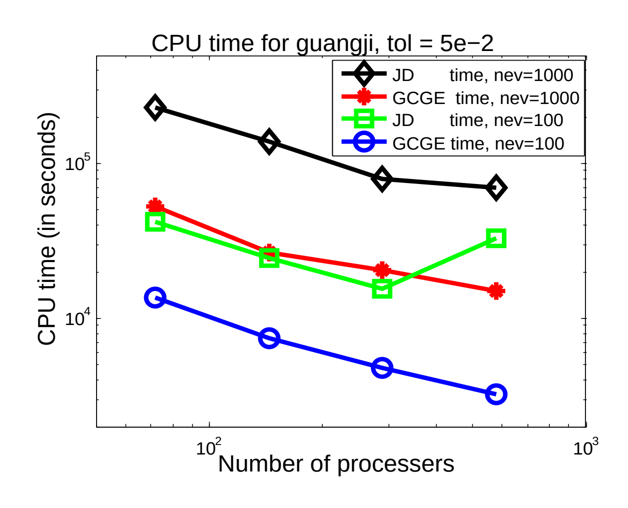


图 8: 2736987阶模态分析矩阵, 收敛准则取5e-2时的计算时间对比.

5 总结

本文基于阻尼块反幂法与子空间投影方法的组合构造了求解特征值问题的广义共轭梯度算法, 并构造了相应的算法软件包GCGE. 该软件包具有稳定、高效、高可拓展性的特点. 针对不同的问题, 计算效率相比于SLEPc软件包中的LOBPCG以及Jacobi-Davidson解法器有2-6倍的效率提升. 且该软件包是Matrix-Free和Vector-Free的, 因此可以有更广泛的应用, 未来可以依据阻尼块反幂法设计求解非对称矩阵特征值问题的相应算法.

参考文献

[1] E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney and D. Sorensen, *LAPACK Users' Guide*, Society for Industrial and Applied Mathematics, 3-rd Version, 1999.

[2] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe and H. van der Vorst, editors, *Templates for the solution of Algebraic Eigenvalue Problems: A Practical Guide*, SIAM, Philadelphia, 2000.

[3] Z. Bai, R. Li and W. Lin, *Linear response eigenvalue problem solved by extended locally optimal preconditioned conjugate gradient methods*, Sci China Math, 59 (2016, 1443-1460.

[4] S. Balay, W. D. Gropp, L. C. McInnes and B. F. Smith, *Efficient Management of Parallelism in Object Oriented Numerical Software Libraries*, in book: Modern Software Tools in Scientific Computing, Edited by E. Arge, A. M. Bruaset and H. P. Langtangen, 163–202, Birkhäuser Press, 1997.

[5] J. A. Duersch, M. Shao, C. Yang and M. Gu, *A robust and efficient implementation of LOBPCG*, SIAM J. Sci. Comput., 40(5) (2018), C655–C676.

[6] 范宣华、陈璞、吴瑞安、肖世富, *基于Jacobi-Davidson算法的大规模模态分析并行计算研究*, 振动与冲击, 33(1) (2014), 203–208.

[7] 范宣华、肖世富、陈璞, *千万自由度量级有限元模态分析并行计算研究*, 振动与冲击, 34(17) (2015), 77-82.

[8] U. Hetmaniuk and R. Lehoucq, *Basis selection in LOBPCG*, J. Comput. Phys., 218 (2006), 324–332.

[9] V. Hernandez, J. E. Roman and V. Vidal, *SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems*, ACM Trans. Math. Software, 31(3) (2005), 351–362.

[10] *HYPRE: High performance preconditioners*, `http://www.llnl.gov/CASC/hypre/`.

[11] A. V. Knyazev, M. E. Argentati, I. Lashuk and E. E. Ovtchinnikov, *Block locally optimal preconditioned eigenvalue xolvers (BLOPEX) in hypre and PETSc*, SIAM J. Sci. Comput., 29(5) (2007), 2224–2239.

[12] A. V. Knyazev, *Toward the optimal preconditioned eigensolver: locally optimal preconditioned conjugate gradient method*, SIAM J. Sci. Comput., 23(2) (2001), 517–541.

[13] A. V. Knyazev and K. Neymeyr, *Efficient soltuion of symmetric eigenvalue problems using multigrid preconditioners in the locally optimal block conjugate gradient method*, Electronic Transactions on Numerical Analysis, 15 (2003), 38–55.

[14] *PASE: Parallel augemented subspace eigensolver*, `https://github.com/pase2017/pase`, 2017.

[15] Y. Saad, *Numerical Methods For Large Eigenvalue Problems*, Second edition, 2011.

[16] A. Stathopoulos and K. Wu, *A block orthogonalization procedure with constant synchronization requirements*, SIAM J. Sci. Comput., 23(6) (2002), 2162–2182.

[17] Z. Wen and Y. Zhang, *Accelerating convergence by augmented Rayleigh-Ritz projections for large-scale eigenvalue compution*, SIAM J. Matrix Anal. Appl., 38(2) (2017), 273-296.

[18] T. Yokozawa, D. Takahashi, T. Boku and M. Sato, *Efficient parallel implementation of classical Gram-Schmidt orthogonalization using matrix multiplication*, In Parallel Matrix Algorithms and Applications (PMAA 2006), pages 37–38, 2006.

[19] 于晨阳、范宣华、王柯颖、肖世富, *基于PANDA平台的多点基础激励谐响应的并行计算*, 计算物理, 35(4) (2018), 443–450.

[20] 张林波、*三维并行自适应有限元软件平台PHG 0.8.6 版: 参考手册、使用指南*, `http://lsec.cc.ac.cn/phg/`, 2012.